Data Science Applications of GPUs in the R Language

Norm Matloff
University of California at Davis

GTC 2016

April 7, 2016

These slides at `http://heather.cs.ucdavis.edu/GTC.pdf`

Why R?

Why R?

- The *lingua franca* for the data science community. (R-Python-Julia battle looming?)
- Statistically Correct: Written by statisticians, for statisticians.
- 8,000 CRAN packages!
- Excellent graphics capabilities, including Shiny (easily build your own interactive tool).

R $\rightarrow$ GPU Link Pros and Cons

R $\rightarrow$ GPU Link Pros and Cons

On the plus side:

- Speed: R is an interpreted language. (Nick Ulle and Duncan Temple Lang working on LLVM compiler.)
- R is often used on large and/or complex data sets, thus requiring large amounts of computation.
- Much of R computation involves matrices or other operations well-suited to GPUs.

On the other hand:

- Big Data implies need for multiple kernel calls, and much host/device traffic.
- Ditto for R's many iterative algorithms.
- Many of the matrix ops are not embarrassingly parallel.
- Unpacking and repacking into R object structure.

Disclaimers

Disclaimers

- Talk is meant to be aimed at NVIDIA but otherwise generic, not focusing on the latest/greatest model.

Disclaimers

- Talk is meant to be aimed at NVIDIA but otherwise generic, not focusing on the latest/greatest model.
- Our running example, NMF, has the goal of illustrating issues and methods concerning the R/GPU interface. It is not claimed to produce the fastest possible computation. (See talk by Wei Tan in this session.)

Running Example: Nonnegative Matrix Factorization (NMF)

Running Example: Nonnegative Matrix Factorization (NMF)

- Have matrix $A \geq 0$, rank r.
- Want to find matrices $W \geq 0$ and $H \geq 0$ of rank $s \ll r$ with

$$A \approx WH$$

- Columns of W form a "pseudo-basis" for columns of A: $A_{.j}$ is approximately a linear combination of the columns of W, with coordinates in $H_{.j}$.

Applications of NMF

Applications of NMF

- Image compression.

Applications of NMF

- Image compression.
- Image classification.

Applications of NMF

- Image compression.
- Image classification. Each column of A is one image.

Applications of NMF

- Image compression.
- Image classification. Each column of A is one image. To classify new image, find coordinates u w.r.t. W, then find nearest neighbor(s) of u in H.

Applications of NMF

- Image compression.
- Image classification. Each column of A is one image. To classify new image, find coordinates u w.r.t. W, then find nearest neighbor(s) of u in H.
- Text classification. Each column of A is one document, with counts of words of interest. Similar to image classification.

Example of R Calling C/C++

Example of R Calling C/C++

- Compare R's **NMF** package to E. Battenberg's **NMF-CUDA**, on a 3430×512 A:

Example of R Calling C/C++

- Compare R's **NMF** package to E. Battenberg's **NMF-CUDA**, on a 3430×512 A:
- R, $s = 10$: 649.843 sec
- GPU, $s = 30$: 0.986 sec
- GPU solved a much bigger problem in much less time
- Even though the R pkg is in C++, not R.

Example of R Calling C/C++

- Compare R's **NMF** package to E. Battenberg's **NMF-CUDA**, on a 3430×512 A:
- R, $s = 10$: 649.843 sec
- GPU, $s = 30$: 0.986 sec
- GPU solved a much bigger problem in much less time
- Even though the R pkg is in C++, not R.
- Solution: Call **NMF-CUDA**'s **update_div()** from R.

Example of R Calling C/C++

- Compare R's **NMF** package to E. Battenberg's **NMF-CUDA**, on a 3430×512 A:
- R, $s = 10$: 649.843 sec
- GPU, $s = 30$: 0.986 sec
- GPU solved a much bigger problem in much less time
- Even though the R pkg is in C++, not R.
- Solution: Call **NMF-CUDA**'s **update_div()** from R. BUT HOW?
- R's **Rcpp** package makes interfacing R to C/C++ very convenient and efficient.

General R/GPU Tools

General R/GPU Tools

What's out there now for R/GPU:

- **gputools**
  (Buckner *et al.*) The oldest major package. Matrix multiply; matrix of distances between rows; linear model fit; QR decomposition; correlation matrix; hierarchical clustering.
- **HiPLAR**
  (Montana *et al.*) R wrapper for **MAGMA** and **PLASMA**. Linear algebra routines, e.g. Cholesky.
- **rpud**
  (Yau.) Similar to **gputools**, but has SVM.
- **Rth**
  (Matloff.) R interfaces to some various algorithms coded in Thrust. Matrix of distances between rows; histogram; column sums; Kendall's Tau; contingency table.

Current Tools (cont'd.)

Current Tools (cont'd.)

- **gmatrix**
  (Morris.) Matrix multiply, matrix subsetting, Kronecker product, row/col sums, Hamiltonian MCMC, Cholesky.
- **RCUDA**
  (Baines and Temple Lang, currently not under active development.) Enables calling GPU kernels directly from R. (Kernels still written in CUDA.)
- **rgpu**
  (Kempenaar, no longer under active development.) "Compiles" simple expressions to GPU.
- various OpenCL interfaces
  ROpenCL, **gpuR**. Similar to **RCUDA**, but via OpenCL interface.

Example: Linear Regression Via gputools

Example: Linear Regression Via gputools

```
> test ← function(n,p) {
   x ← matrix(runif(n*p),nrow=n)
   regvals ← x %*% rep(1.0,p)
   y ← regvals + 0.2*runif(n)
   xy ← cbind(x,y)
   print("gputools method")
   print(system.time(gpuLm.fit(x,y)))
   print("ordinary method")
   print(system.time(lm.fit(x,y)))
}
> test(100000,1500)
[1] "gputools method"
   user  system elapsed
  6.280   2.878  17.902
[1] "ordinary method"
   user  system elapsed
142.282   0.669 142.912
```

Key Issue: Keeping Objects on the Device

Key Issue: Keeping Objects on the Device

- Some packages, notably **gputools**, do not take arguments on the device.
- So, cannot store intermediate results on the device, thus requiring needless copying.
- Some packages remedy this, e.g. **gmatrix**.

Example

Example

```
library(gputools)
library(gmatrix)
n ← 5000
z ← matrix(runif(n^2),nrow=n)
# plain R:
system.time(z %*% z %*% z)
#     user  system  elapsed
# 138.757   0.322  139.081
system.time(gpuMatMult(gpuMatMult(z,z),z))
#     user  system  elapsed
#    6.607   1.170   10.059
zm ← gmatrix(z,nrow=n,ncol=n) # zm2, zm3 not shown
system.time({gmm(zm,zm,zm2); gmm(zm,zm2,zm3)})
#     user  system  elapsed
#    6.258   1.031    7.285
```

Rth Example — Kendall's Tau

Rth Example — Kendall's Tau

A kind of correlation measure, defined to be the proportion of *concordant pairs*:
(X_i, Y_i) and (X_j, Y_j) are concordant if
$sign(X_i - X_j) \cdot sign(Y_i - Y_j) > 0$

Kendall's Tau (cont'd.)

Kendall's Tau (cont'd.)

R wrapper to Thrust call:

```
rthkendall <- function(x,y) {
   dyn.load("rthkendall.so")
   n <- length(x)
   tmp <-
      .C("rthkendall",as.single(x),as.single(y),
      as.integer(n),tmpres=single(1),DUP=dupval)
   return(tmp$tmpres)
}
```

Kendall's Tau (cont'd)

Kendall's Tau (cont'd)

```
void rthkendall(float *x, float *y,
   int *nptr, float *tauptr)
{  int n = *nptr;
   thrust::counting_iterator<int> seqa(0);
   thrust::counting_iterator<int> seqb = seqa + n-1;
   // dx, dy, tmp declarations not shown
   thrust::transform(seqa,seqb,tmp.begin(),
      calcgti(dx,dy,n));
   int totcount =
      thrust::reduce(tmp.begin(),tmp.end());
   float npairs = n * (n-1) / 2;
   *tauptr = (totcount - (npairs-totcount)) / npairs
}
```

Kendall's Tau (cont'd)

Kendall's Tau (cont'd)

```
struct calcgti {  // handle 1 i, all j > i
   // more declarations not shown
   calcgti(floublevec _dx,floublevec _dy,int _n) :
      dx(_dx),
      dy(_dy),
      n(_n)
      {  wdx = thrust::raw_pointer_cast(&dx[0]);
         wdy = thrust::raw_pointer_cast(&dy[0]);
}
   __device__ int operator()(int i)
   {  flouble xi = wdx[i], yi = wdy[i];
      int j,count=0;
      for (j = i+1; j < n; j++)
         count +=
            ( (xi - wdx[j]) * (yi - wdy[j]) > 0);
      return count;
   }
};
```

Example: NMF Again

Example: NMF Again

- The R **NMF** package, and **NMF-CUDA** use *multiplicative update* methods.
- For instance, for Frobenius norm,

$$W \leftarrow W \circ \frac{AH'}{WHH'}$$

and similarly for H.

Example: NMF Again

- The R **NMF** package, and **NMF-CUDA** use *multiplicative update* methods.
- For instance, for Frobenius norm,

$$W \leftarrow W \circ \frac{AH'}{WHH'}$$

  and similarly for H.
- Another possibility is to use the *alternating least squares* method:
  - In odd-numbered iterations, regress each col. of A against cols. of W, yielding the columns of H. Mult. update even better suited to GPUs.
  - In even-numbered iterations, reverse the roles of W and H (and now with rows).
- As seen earlier, least-squares estimation can be done fairly well on GPUs.

RCUDA Example: Normal Density

RCUDA Example: Normal Density

Basic goal: Call CUDA kernels from R without burdening the R programmer with details of configuring grids, allocating device memory, copying between host and device, etc.

Kernel:

```
extern "C"
__global__ void
  dnorm_kernel(float *vals, int n, float mu, float sig)
{
  int myblock = blockIdx.x + blockIdx.y * gridDim.x;
  int blocksize =
     blockDim.x * blockDim.y * blockDim.z;
  int subthread =
     threadIdx.z*(blockDim.x * blockDim.y) +
     threadIdx.y*blockDim.x + threadIdx.x;
  int idx = myblock * blocksize + subthread
  float std = (vals[idx] - mu)/sig;
  float e = exp( - 0.5 * std * std);
  vals[idx] = e / ( sig * sqrt(2 * 3.14159));
}
```

RCUDA (cont'd.)

RCUDA (cont'd.)

```
n = 1e6
mean = 2.3
sd = 2.1
x = rnorm(n, mean, sd)
# eval density at all pts in x
m = loadModule("dnorm.ptx")
k = m$dnorm_kernel
ans = .cuda(k,x,n,mean,sd,
   gridDim = c(62, 32),blockDim = 512)
```

Helpful Utilities

Helpful Utilities

- **Rcpp**
  - Greatly facilitates calling C/C++ from R.
  - Base R offers functions **.C()** and **.Call()**. The former is inefficient and the latter requires knowledge of R internals.
  - **Rcpp** makes it easy.
- **bigmemory**
  - R currently not completely 64-bit.
  - Can have 52-bit integers, but only 32-bit matrix row/col dimensions.
  - The **bigmemory** package allows storing R matrices in "C land," circumventing R storage limits.
  - Storage is in **shmem**, thus allowing for multicore use **Rdsm**).

Software Alchemy

Software Alchemy

- For “statistical” problems, in “iid” form.

Software Alchemy

- For “statistical” problems, in “iid” form. Image, text classification work.

Software Alchemy

- For “statistical” problems, in “iid” form. Image, text classification work.
- Simple idea:
  - Break data into “independent” chunks.
  - Apply the procedure, e.g. logistic regression, to each chunk.
  - Use combining op, e.g. averaging, for final answer.
  - Provably correct and efficient.

Software Alchemy

- For "statistical" problems, in "iid" form. Image, text classification work.
- Simple idea:
  - Break data into "independent" chunks.
  - Apply the procedure, e.g. logistic regression, to each chunk.
  - Use combining op, e.g. averaging, for final answer.
  - Provably correct and efficient.
- A variant: Apply procedure to chunks but take combining op to be concatenation them rather than averaging.

Serial Benefits of Software Alchemy

Serial Benefits of Software Alchemy

- SA gives speedup even in serial case of task is $O(n^c)$ for $c > 1$
- Use SA to address a common problem: Big data, small GPU memory.

Serial Benefits of Software Alchemy

- SA gives speedup even in serial case of task is $O(n^c)$ for $c > 1$
- Use SA to address a common problem: Big data, small GPU memory. Apply GPU to each chunk, serially, then run combining op.

Serial Benefits of Software Alchemy

- SA gives speedup even in serial case of task is $O(n^c)$ for $c > 1$
- Use SA to address a common problem: Big data, small GPU memory. Apply GPU to each chunk, serially, then run combining op.

Example: NMF

Example: NMF

- E.g. break rows or columsn into m chunks.
- Get approximation WH for each one.
- To predict new case:
  - Get the m predictions.
  - Combine via voting.